```cpp
#include "GameplayState.h"

#include <conio.h>
#include <iostream>
#include <assert.h>

#include "Enemy.h"
#include "Key.h"
#include "Door.h"
#include "Money.h"
#include "Goal.h"
#include "AudioManager.h"
#include "Game.h"
#include "Utility.h"

#include "StateMachineExampleGame.h"

using namespace std;

constexpr int kArrowInput = 224;
constexpr int kLeftArrow = 75;
constexpr int kRightArrow = 77;
constexpr int kUpArrow = 72;
constexpr int kDownArrow = 80;
constexpr int kEscapeKey = 27;
constexpr int kBackspace = 8;

GameplayState::GameplayState(StateMachineExampleGame* pOwner)
    : m_pOwner(pOwner)
    , m_beatLevel(false)
    , m_skipFrameCount(0)
    , m_currentLevel(0)
    , m_pLevel(nullptr)
{
    m_LevelNames.push_back("Level4.txt");
    m_LevelNames.push_back("Level5.txt");
    m_LevelNames.push_back("Level6.txt");
}

GameplayState::~GameplayState()
{
    m_pLevel = nullptr;
    delete m_pLevel;
}

bool GameplayState::load()
{
    if (m_pLevel)
    {
        delete m_pLevel;
        m_pLevel = nullptr;
    }

```

```
54      m_pLevel = new Level();
55
56      return m_pLevel->LoadLevel(m_LevelNames.at(m_currentLevel),
          m_player.GetXPositionPointer(), m_player.GetYPositionPointer());
57  }
58
59  void GameplayState::Enter()
60  {
61      load();
62  }
63
64  bool GameplayState::Update(bool processInput)
65  {
66      if (processInput && !m_beatLevel)
67      {
68          int input = _getch();
69          int arrowInput = 0;
70          int newPlayerX = m_player.GetXPosition();
71          int newPlayerY = m_player.GetYPosition();
72
73          // One of the Arrow keys were pressed
74          if (input == kArrowInput)
75          {
76              arrowInput = _getch();
77          }
78
79          if ((input == kArrowInput && arrowInput == kRightArrow) ||
80              ((char)input == 'd' || (char)input == 'D'))
81          {
82              newPlayerX++;
83          }
84
85          else if ((input == kArrowInput && arrowInput == kLeftArrow) ||
86              ((char)input == 'a' || (char)input == 'A'))
87          {
88              newPlayerX--;
89          }
90
91          else if ((input == kArrowInput && arrowInput == kUpArrow) ||
92              ((char)input == 'w' || (char)input == 'W'))
93          {
94              newPlayerY--;
95          }
96
97          else if ((input == kArrowInput && arrowInput == kDownArrow) ||
98              ((char)input == 's' || (char)input == 'S'))
99          {
100             newPlayerY++;
101         }
102
103         else if (input == kEscapeKey)
104         {
105             m_pOwner->LoadScene
```

```
                (StateMachineExampleGame::SceneName::MainMenu);
106         }
107         else if ((char)input == 'Z' || (char)input == 'z')
108         {
109             m_player.DropKey();
110             AudioManager::GetInstance()->dropKeySound();
111         }
112         //If position never changed
113
114         if (newPlayerX == m_player.GetXPosition() && newPlayerY ==
              m_player.GetYPosition())
115         {
116
117         }
118         else
119         {
120             HandleCollision(newPlayerX, newPlayerY);
121         }
122     }
123
124     if (m_beatLevel)
125     {
126         ++m_skipFrameCount;
127         if (m_skipFrameCount > kFramesToSkip) // player transitions over to
              X spot before sound.
128         {
129             m_beatLevel = false;
130             m_skipFrameCount = 0;
131
132             ++m_currentLevel;
133             if (m_currentLevel == m_LevelNames.size())
134             {
135                 Utility::WriteHighScore(m_player.GetMoney());
136                 AudioManager::GetInstance()->win();
137                 m_pOwner->LoadScene
                      (StateMachineExampleGame::SceneName::Win);
138             }
139             else
140             {
141                 load();
142             }
143         }
144     }
145     return false;
146 }
147
148 void GameplayState::HandleCollision(int newPlayerX, int newPlayerY) // more
      parameters to help with if loop
149 {
150     bool isGameDone = false;
151     PlaceableActor* collidedActor = m_pLevel->UpdateActors(newPlayerX,
          newPlayerY); // creates a placeable actor
152     if (collidedActor != nullptr && collidedActor->IsActive())
```

```
153     {
154         switch (collidedActor->GetType())
155         {
156         case ActorType::Enemy:
157         {
158             Enemy* collidedEnemy = dynamic_cast<Enemy*>(collidedActor); //
                  specifies the type/ thing we are trying to cast, in this case
                  an enermy
159             assert(collidedEnemy);
160             AudioManager::GetInstance()->loseLife();
161             // if the pointer is valid, if statement works, if it is a key
                  none of the code will work
162             collidedEnemy->Remove(); // if a collision with an enemy occurs,
                   the enermy is removed.
163             m_player.SetXYPosition(newPlayerX, newPlayerY); // players
                  position is set to new position
164             m_player.DecrementLives(); // decrmeent lives
165             if (m_player.GetLive() < 0) // if less than zero game is over.
166             {
167                 AudioManager::GetInstance()->PlayLoseSound();
168                 m_pOwner->LoadScene
                      (StateMachineExampleGame::SceneName::Lose);
169             }
170             break;
171         }
172         case ActorType::Money:
173         {
174             Money* collidedMoney = dynamic_cast<Money*>(collidedActor); //
                  if collided with money
175             assert(collidedMoney);
176             AudioManager::GetInstance()->moneySound();
177             collidedMoney->Remove(); // remove the money
178             m_player.AddMoney(collidedMoney->GetWorth()); // add the money
                  and show the worth.
179             m_player.SetXYPosition(newPlayerX, newPlayerY);
180             break;
181         }
182         case ActorType::Key:
183         {
184             Key* collidedKey = dynamic_cast<Key*>(collidedActor); //
                  returning null if fails within dynamic casts.
185             assert(collidedKey);
186             if (!m_player.HasKey())
187             {
188                 m_player.PickUpKey(collidedKey);
189                 AudioManager::GetInstance()->pickupkey();
190                 collidedKey->Remove();
191                 m_player.SetXYPosition(newPlayerX, newPlayerY);
192             }
193             break;
194         }
195         case ActorType::Door:
196         {
```

```
197             Door* collidedDoor = dynamic_cast<Door*>(collidedActor);
198             assert(collidedDoor);
199             if (!collidedDoor->IsOpen())
200             {
201                 if (m_player.HasKey(collidedDoor->GetColour()))
202                 {
203                     collidedDoor->Open();
204                     collidedDoor->Remove();
205                     m_player.UseKey();
206                     m_player.SetXYPosition(newPlayerX, newPlayerY);
207                     AudioManager::GetInstance()->dropKeySound();
208                 }
209                 else
210                 {
211 
212                 }
213             }
214             else
215             {
216                 m_player.SetXYPosition(newPlayerX, newPlayerY); // player
                      goes through the door
217             }
218             break;
219         }
220         case ActorType::Goal:
221         {
222             Goal* collidedGoal = dynamic_cast<Goal*>(collidedActor);
223             assert(collidedGoal);
224             collidedGoal->Remove(); // removes actors
225             m_player.SetXYPosition(newPlayerX, newPlayerY);
226             m_beatLevel = true;
227             break;
228         }
229         }
230     }
231     else if (m_pLevel->IsSpace(newPlayerX, newPlayerY)) // no collision
232     {
233         m_player.SetXYPosition(newPlayerX, newPlayerY);
234     }
235     else if (m_pLevel->IsWall(newPlayerX, newPlayerY))
236     {
237         // wall collision
238     }
239 }
240 
241 void GameplayState::Draw()
242 {
243     HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
244     system("cls");
245 
246     m_pLevel->Draw();
247 
248     //Set cursor position for player
```

```
249     COORD actorCursorPosition;
250     actorCursorPosition.X = m_player.GetXPosition();
251     actorCursorPosition.Y = m_player.GetYPosition();
252     SetConsoleCursorPosition(console, actorCursorPosition);
253     m_player.Draw();
254
255
256     //Set cursor to end of level.
257     COORD currentCursorPosition;
258     actorCursorPosition.X = 0;
259     actorCursorPosition.Y = m_pLevel->GetHeight();
260     SetConsoleCursorPosition(console, actorCursorPosition);
261
262     DrawHUD(console);
263 }
264
265 void GameplayState::DrawHUD(const HANDLE& console)
266 {
267     cout << endl;
268
269     // Top Border
270     for (int i = 0; i < m_pLevel->GetWidth(); ++i)
271     {
272         cout << Level::WAL;
273     }
274     cout << endl;
275
276     // left border
277
278     cout << Level::WAL;
279
280     cout << " wasd - move " << Level::WAL << " z - drop key " << Level::WAL;
281     cout << "$: " << m_player.GetMoney() << " " << Level::WAL;
282     cout << "Lives: " << m_player.GetLive() << " " << Level::WAL;
283     cout << "Key: ";
284     if (m_player.HasKey())
285     {
286         m_player.GetKey()->Draw();
287     }
288     else
289     {
290         cout << " ";
291     }
292
293     // right border
294
295     CONSOLE_SCREEN_BUFFER_INFO csbi;
296     GetConsoleScreenBufferInfo(console, &csbi);
297
298     COORD pos;
299     pos.X = m_pLevel->GetWidth() - 1;
300     pos.Y = csbi.dwCursorPosition.Y;
301     SetConsoleCursorPosition(console, pos);
```

```
302
303     cout << Level::WAL;
304     cout << endl;
305
306     // Bottom Border
307     for (int i = 0; i < m_pLevel->GetWidth(); ++i)
308     {
309         cout << Level::WAL;
310     }
311     cout << endl;
312 }
```